UniSenac - Campus Pelotas
Prof. Pablo De Chiaro Rosa

Descrição da Atividade

Esta atividade tem como objetivo implementar e comparar o desempenho de duas estruturas de dados fundamentais: **Listas** e **Árvores Binárias**. Vocês deverão desenvolver implementações próprias dessas estruturas e realizar uma análise experimental detalhada sobre seus comportamentos com diferentes volumes de dados.

Conjuntos de Dados para Experimentação

Para os experimentos, utilizem os seguintes conjuntos de valores fornecidos:

Arquivos de Dados:

- **Conjunto Pequeno**: conjunto_pequeno.txt - 100.000 valores (684KB)
    - Download: Link do Conjunto Pequeno
- **Conjunto Médio**: conjunto_medio.txt - 5.000.000 valores (43MB)
    - Download: Link do Conjunto Médio
- **Conjunto Grande**: conjunto_grande.txt - 30.000.000 valores (284MB)
    - Download: Link do Conjunto Grande

Requisitos Técnicos Obrigatórios

1. Implementação da Classe Lista

- Desenvolver uma classe que opere exclusivamente com **LISTA**
- Implementar método para **inserção** de elementos
- Implementar método para **busca** de elementos
- **Obrigatório**: A lista deve iniciar **VAZIA**

2. Implementação da Classe Árvore Binária

- Desenvolver uma classe que opere exclusivamente com **ÁRVORE BINÁRIA**
- Implementar método para **inserção** de elementos
- Implementar método para **busca** de elementos
- **Obrigatório**: A árvore deve iniciar **VAZIA**
- **Base de Desenvolvimento**: Utilize como ponto de partida o exemplo apresentado na aula de 23/09, aplicando princípios de orientação a objetos

3. Código-fonte de Referência

- Material de apoio disponível em: Código-fonte de Referência

4. Leitura de Arquivos

```python
def ler_arquivo(nome_arquivo):
    """
    Lê um arquivo txt e retorna os valores como uma lista de inteiros.

    Parâmetros:
    - nome_arquivo: nome do arquivo txt a ser lido

    Retorna:
    - lista com os valores inteiros do arquivo
    """
    valores = []
    with open(nome_arquivo, "r") as arquivo:
        for linha in arquivo:
            valor = int(linha.strip())
            valores.append(valor)
    return valores
```

Questões de Pesquisa e Desenvolvimento

Questão 1: Ponto de Eficiência

Objetivo: Determinar o ponto de virada onde árvores binárias se tornam mais eficientes que listas.

Tarefa: A partir de quantos valores (n) se mostra mais eficiente buscar elementos utilizando uma estrutura de árvore binária ao invés de uma lista? **Prove em código** sua resposta.

Questão 2: Análise Temporal Detalhada

Objetivo: Medir e comparar tempos de execução para diferentes operações.

Tarefa: Documente os tempos parciais para cada conjunto de dados considerando:

- Tempo para **construir** cada estrutura
- Tempo para **buscar n valores** que **existem** em cada estrutura
- Tempo para **buscar n valores** que **NÃO existem** em cada estrutura
- Realize **múltiplos experimentos** e calcule tempos agregados para cada configuração (`time.perf_counter()`)

Questão 3: Análise de Ineficiências

Objetivo: Investigar limitações das listas em grandes volumes de dados.

Tarefa: As listas se mostram mais lentas para grandes conjuntos durante a busca por valores? Se são ineficientes, explique **detalhadamente** os motivos técnicos dessa ineficiência.

Questão 4: Otimização de Listas

Objetivo: Explorar possibilidades de melhoria mantendo a estrutura de lista.

Tarefa: É possível otimizar o algoritmo da Classe Lista criada, sem deixar de operar em uma LISTA, para melhorar a performance de busca? Considere implementar:

- Algoritmos de ordenação (merge sort, quick sort, etc.)
- Algoritmos de busca otimizada (busca binária)
- Outras técnicas de otimização

Prove a eficácia das otimizações implementadas.

Restrições Importantes

Proibições Absolutas:

- **NÃO** utilizar funções prontas (built-in functions) do Python
- **Exemplos proibidos**: sort(), set(), min(), max(), sum(), entre outras
- **Regra**: Para toda função necessária, você deve implementá-la manualmente
- **Objetivo**: Demonstrar compreensão completa dos algoritmos fundamentais

Entregáveis Obrigatórios

1. Arquivo Python (.py)

- **Arquivo único** contendo todas as classes implementadas
- Documentação clara do código com comentários explicativos
- Exemplos de uso das classes desenvolvidas
- Código organizado e seguindo boas práticas de programação

2. Relatório Técnico (PDF)

- **Explicação detalhada** das decisões algorítmicas tomadas
- **Justificativa** para a criação de cada classe e método
- **Análise** dos problemas enfrentados e soluções implementadas
- **Discussão** sobre as vantagens e desvantagens de cada estrutura

Prazo de Entrega

UniSenac - Campus Pelotas

Prof. Pablo De Chiaro Rosa

Data limite: 30 de setembro de 2025, às 23:59 **Local de entrega**: Atividade correspondente na seção "Conteúdos" do Blackboard

Dicas para o Desenvolvimento

Organização do Código:

- Mantenha classes bem estruturadas com métodos claramente definidos
- Use comentários explicativos para algoritmos complexos
- Implemente tratamento de erros adequado
- Teste suas implementações com dados pequenos antes de usar os conjuntos grandes

Para o Relatório:

- Inclua gráficos de desempenho comparativo
- Documente todos os experimentos realizados
- Explique escolhas de implementação de forma técnica
- Apresente conclusões bem fundamentadas

Experimentação:

- Realize múltiplas execuções para obter médias confiáveis
- Varie o tamanho dos dados para identificar padrões
- Documente configurações do sistema utilizado nos testes
- Compare não apenas tempos, mas também uso de memória quando relevante

Critérios de Avaliação

- **Correção técnica** das implementações (50%)
- **Completude** dos experimentos realizados (25%)
- **Qualidade** do relatório e **análise** dos resultados (25%)

Esta atividade é uma oportunidade de aprofundar conhecimentos sobre estruturas de dados fundamentais e desenvolver habilidades de análise experimental. Dediquem tempo adequado tanto à implementação quanto à análise dos resultados.

Que a força esteja com você!